(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
23 August 2001 (23.08.2001)

PCT

(10) International Publication Number
WO 01/61916 A2

(51) International Patent Classification[7]: H04L 9/06

(21) International Application Number: PCT/CA01/00201

(22) International Filing Date: 19 February 2001 (19.02.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
2,298,990 18 February 2000 (18.02.2000) CA

(71) Applicant *(for all designated States except US)*: **CLOAKWARE CORPORATION** [CA/CA]; Suite 311, 260 Hearst Way, Kanata, Ontario K2L 3H1 (CA).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **CHOW, Stanley, T.** [CA/CA]; 3338 Carling Avenue, Nepean, Ontario K2H 5A8 (CA). **JOHNSON, Harold, J.** [CA/CA]; 4 Floral Place, Nepean, Ontario K2H 6N7 (CA). **XIAO, James, Zhengchu** [CA/CA]; 118 Goldridge Drive, Kanata, Ontario K2T 1G3 (CA).

(74) Agents: **O'NEILL, Gary** et al.; Gowling Lafleur Henderson LLP, Suite 2600, 160 Elgin Street, Ottawa, Ontario K1P 1C3 (CA).

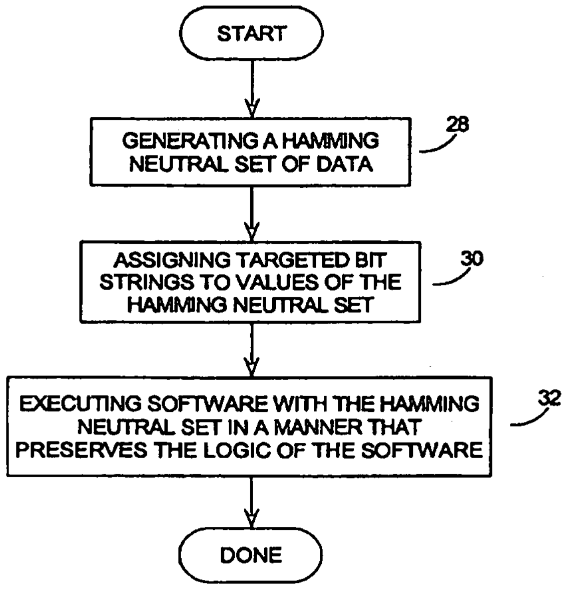
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: ENCODING METHOD AND SYSTEM RESISTANT TO POWER ANALYSIS


START
GENERATING A HAMMING NEUTRAL SET OF DATA
28
ASSIGNING TARGETED BIT STRINGS TO VALUES OF THE HAMMING NEUTRAL SET
30
EXECUTING SOFTWARE WITH THE HAMMING NEUTRAL SET IN A MANNER THAT PRESERVES THE LOGIC OF THE SOFTWARE
32
DONE

(57) Abstract: New techniques for cracking sealed platforms have recently been discovered which observe power modulation during execution of a software encryption program on a computer processor. Particularly vulnerable to such simple power analysis and differential power analysis attacks are smart cards which employ Data Encryption Standard (DES) protection. The invention protects against such attacks by mapping data onto "Hamming-neutral" values, that is, bytes which have the same number of 1-values, so power signatures do not vary during execution. The Hamming-neutral values are assigned to each bit-string in a targeted data set, rather than in a bit-wise manner as known. This approach has a number of advantages: it is less demanding of system resources, it results in a larger number of encodings for an attacker to decipher, and it can be applied to various components including: addressing, indexing, stored data and input data. Many variations and improvements are also described.

WO 01/61916 A2

Encoding Method and System Resistant to Power Analysis

The present invention relates generally to computer software and electronic hardware, and more specifically, to a method, apparatus and system resistant to power analysis of sealed platforms, including a particular implementation for smart cards employing Data Encryption Standard (DES) protection.

Background of the Invention

Keeping electronic information hidden from hostile parties is desirable in many environments, whether personal, business, government, or military. Recently, "sealed platforms", which are special kinds of electronic hardware devices, have been developed to satisfy this need. The term "platform" generally refers to a hardware/software environment capable of supporting computation including the execution of software programs. A "sealed" platform refers to a platform purposely built to frustrate reverse-engineering.

In contrast to traditional credit and debit cards which store a small amount of information on a magnetic strip, the new sealed platforms such as smart cards, may store and process a significantly larger quantity of data using microprocessors, random access memory (RAM), and read only memory (ROM). The new sealed platforms are typically secured using cryptographic technology which is intended to maintain and manipulate secret parameters in open environments without revealing their values. Compromise of a secret key used to compute a digital signature could, for example, allow an attacker to forge the owner's digital signature and execute fraudulent transactions.

A sealed platform is intended to perform its function while protecting information and algorithms, such as performing digital signatures as part of a challenge-response protocol, authenticating commands or requests, and encrypting or decrypting arbitrary data. A smart card used in a stored value system may, for example, digitally sign or compute parameters such as the smart card's serial number, account balance, expiration date, transaction counter, currency, and transaction amount as part of a value transfer.

Figure 1 presents an exemplary physical structure of a smart card **10**, which typically embeds an electronic chip **12** or chips in a plastic card **14**. The electronic chip **12** may include, for example, a microprocessor or similar device, read-only memory (ROM), and/or read-write random access memory (RAM). The electronic chip **12** may also include other electronic components such as digital signal processors (DSPs), field-programmable gate arrays (FPGAs), electrically-erasable programmable read-only memory (EEPROM) and miscellaneous support logic.

Generally, the electronic chip **12** is glued into a recessed area **16** of the plastic card **14** and is covered by a printed circuit **18** which provides the electrical interface to an external smart card reader. The standard configuration of the input and output pads of the printed circuit **18** is shown in detail in **Figure 1**, and generally includes power (VCC), ground (GND), a clock input (CLK) and a serial input/output pad (I/O). Several additional unconnected pads (N/C) are also included in the standard configuration. Because the plastic card **14** is somewhat flexible, the electronic chip **12** must be small enough to avoid breaking. This limits the physical size of the electronic chip **12** to a few millimetres across, and also limits the number of electronic components that can be supported.

Contactless smart cards are also in use, which communicate with the external smart card reader using radio frequencies or other wireless communication media. Such smart cards are generally equipped with an internal antenna, rather than the input and output pads of the printed circuit **18**.

Data Encryption Standard

Smart cards commonly encode their internal data using a cryptographic technique such as the Data Encryption Standard (DES). DES is a block cipher method using a 64 bit key (of which only 56 bits are actually used), which is very fast and has been widely adopted. Though DES can be cracked by a brute-force attack (simply testing all possible keys), triple DES is still considered very secure (triple DES is simply three copies of DES executed in series).

For the purposes of the examples described hereinafter, it is sufficient to know that the DES algorithm performs 16 rounds which effect lookups to eight separate translation tables called S-boxes. A detailed description of the DES is beyond the scope of this discussion, but is presented by Bruce Schneier in *Applied Cryptography*, 2nd edition, ISBN 0-471-11709-9, 1996, John Wiley & Sons, at pp. 265-294. For the Federal Information Processing Standard (FIPS) description of DES, see FIPS publication 46-3, available on the Internet at http://csrc.nist.gov/fips/.

Other similar cryptographic techniques are also known in the art, including: triple DES, IDEA, SEAL, and RC4; public key (asymmetric) encryption and decryption using RSA and ElGamal; digital signatures using DSA, ElGamal, and

RSA; and Diffie-Hellman key agreement protocols. Despite the theoretical strength and complexity of these cryptographic systems, Power Analysis techniques have recently been developed which allow these keys to be cracked quite quickly.

Power Analysis (PA)

Power analysis is the process of gathering information about the data and algorithms embodied on a platform by means of the "power signature" of the platform. The "power signature" of a platform is its power consumption profile measured over time, while executing the software stored on that platform.

The power consumed by a microprocessor, micro-controller or similar electronic device changes with the state of the electronic components in the device. Such devices generally represent data in terms of binary 1s and 0s, which are represented in the electronic devices as corresponding high or low voltage levels. For example a value of 1 may be represented by +5 volts and a value of 0 by 0 volts.

Hence, the amount of power that a sealed platform consumes may be correlated with the number of binary 1s in a data word, at a given moment in time. It follows that the amount of current drawn by, and the electromagnetic radiation emanated from a sealed platform, may be correlated to the secrets being manipulated within it. Such signals can be measured and analysed by attackers to recover secret keys.

State transitions are also a major influence on the power consumption of a device performing a computation. As the value of a bit changes, transistor switches associated with that bit change state. Therefore, there is an increase in the amount of power consumed when the system is in transition.

Paul Kocher, Joshua Jaffe and Benjamin Jun, in their paper: *Introduction to differential power analysis and related attacks*, 1998 (available on the Internet at http://www.cryptography.com/dpa/technical), show that attackers can often non-invasively extract secret keys using external measurement and analysis of a device's power consumption, electromagnetic radiation, or processor cycle timing during performance of cryptographic operations. Other similar extraction techniques would be clear to one skilled in the art from the teachings of Kocher et al.

Smart cards, for example, require an external power supply to operate. The current and voltage being supplied to the smart card may easily be monitored while it is executing, using an arrangement such as that presented in **Figure 2**. In this arrangement, the smart card **10** is provided with an external power supply unit (PSU)

20, and its operation is monitored using a standard personal computer **22** running appropriate analysis software. The power consumed by the smart card **10** is monitored using a pickup **24**, whose data is digitized for the personal computer (PC) **22** using an analogue to digital convertor (A/D) **26**. The PC **22** also provides a clock signal (CLK) to the smart card **10** and communicates data via its serial input and output port (DIGITAL I/O). This arrangement allows the attacker to monitor the power consumed by the smart card **10** while it is processing known data.

Simple Power Analysis (SPA)

In simple power analysis (SPA), the power signature for the execution of a given algorithm is used to determine information about the algorithm and its data. Generally, power data is gathered from many executions and averaged at each point in time in the profile.

For example, if SPA is used to attack a DES key space, and the attacker has access to the specific code, but not the particular DES key, a particular series of points in the power signature may indicate the number of 1s and 0s in each 8-bit byte of the DES key (note that the term "byte" will generally refer to an 8-bit byte in this document). This reduces the space of possible keys for an exhaustive all-possible-keys attack from 2^{56} possible keys to 2^{38} possible keys (if parity bits are stored for each byte of the key), making search time among possible keys about 2^{18} times shorter.

Differential Power Analysis (DPA)

Differential power analysis (DPA) is a form of power analysis in which information is extracted by means of gathering multiple power signatures and analysing the differences between them (see Paul Kocher, Joshua Jaffe and Benjamin Jun, 1998, *Introduction to differential power analysis and related attacks;* available at http://www.cryptography.com/dpa/technical). For certain kinds of data and algorithms exhibiting repetitious behaviour, it is an extraordinarily effective method for penetrating secrets stored on sealed platforms. It can reveal information about the data resulting from computations, fetches from memory, stores to memory, the data addresses in the memory of the sealed platform from which data are fetched or to which data are stored during execution, and the code addresses from which instructions are fetched during the execution of algorithms on the sealed platform. These capabilities render protection of sealed platforms against DPA attack both very important to security and very difficult to achieve on inexpensive sealed platforms.

While SPA attacks use primarily visual inspection to identify relevant power fluctuations, DPA attacks use statistical analysis and error correction techniques to extract information correlated to secret keys. Hence, DPA is a much more powerful attack than SPA, and is much more difficult to prevent.

One use for DPA is to extract cryptographic keys for encryption or decryption performed on a sealed platform. For the Data Encryption Standard (DES), DPA has proved extremely effective; low-cost smart cards performing DES have proven, in recent experience, to be highly vulnerable to DPA. Any form of encryption or decryption which is similar to DES would necessarily have similar vulnerabilities when incarnated on low-cost smart cards or similar sealed platforms.

DPA Example: Finding a DES Key

Implementation of a DPA attack involves two phases: data collection, followed by data analysis. Data collection for DPA may be performed as described with respect to **Figure 2**, by sampling a device's power consumption during cryptographic operations as a function of time or number of clock cycles. For DPA, a number of cryptographic operations using the target key are observed.

To perform such an attack on a smart card, one processes a large number (a thousand or more) DES encryptions (or decryptions) on distinct plaintexts (or cyphertexts), recording:

1. the power profile;
2. the input, chosen at random by the attacker; and
3. the output, computed by the smart card as the encrypted of decrypted value with the hidden key for each.

Each power profile is referred to as a sample.

In each round of DES, the output of a given S-box is dependent on both the data to be encrypted (or decrypted) and the key. Since the attacker knows the input text, he guesses what the value of the key is, that was used to generate a particular power signature sample, so he can determine whether a particular output bit of a given S-box is 1 or 0 for the particular data used in the sample (note that each standard S-box has a 6-bit input and a 4-bit output). Typically, this analysis begins in round 1 or 16 since those are the ones where the attacker knows either the exact inputs (for round 1) or outputs (for round 16) for the respective S-box.

The attacker does not know the key, but because the DES algorithm only performs one S-box lookup at a time, it is only necessary to guess the six bits of the secret key that are relevant to the S-box being observed (and corresponding to the power consumption) at that time. As only 6-bits are relevant, it is only necessary to test $2^6 = 64$ possible sequences of values for a given 6-bit portion of the 56-bit secret key. For each guess of the values of these six bits, one divides the samples into two groups: those in which the targeted output bit (that is, one of the four output bits from a targeted S-box which is chosen as a target in the first round of the attack) is a 1 if the attacker's guess of the six key bits is correct (the 1-group), and those in which it is a 0 if the attacker's guess of the six key bits is correct (the 0-group).

The power samples in each group are then averaged. On average, modulo minor asymmetries in DES, those portions of the averaged power profiles which are affected only by bits other than the particular output bit mentioned above, should be similar, since on average, in both groups, they should be 1 for about half of the samples in each group, and 0 for about half of the samples in each group.

However, those portions of the averaged power profiles which are affected by the above-mentioned output bit should show a distinct difference between the 1-group and the 0-group. The presence of such a difference, or multiple such differences, indicates that the guessed value of the six key bits was correct. Its absence, or the absence of such differences, shows that the guessed value of the six key bits was incorrect.

This process of guessing at the value of the secret key, dividing the power signature samples into those which will yield a 1-output and those which will yield a 0-output (the 1-group and 0-group respectively), averaging the profiles, and seeking the above-mentioned distinct difference, is repeated until a guess is shown to be correct. One then has six bits of the key.

The above guessing procedure is repeated for the other seven S-boxes. When all S-boxes have been treated in this way, one has obtained 48 out of the 56 key bits, leaving only eight bits undetermined. This leaves a remaining space of $2^8 = 256$ possible keys to find the balance of the correct secret key.

Note how little information the attacker needs to employ such an attack. The attacker does not have to know:

1. the specific code used to implement DES;
2. the memory layout used for storing the S-boxes;

3. where in the power profile the distinct difference or difference, if any, is expected to appear for a correct guess;
4. how many such distinct differences are expected to appear in the power profile for a correct guess; or
5. whether the chosen S-box output bits are normal or complemented as flipping 1s and 0s will produce the same kind of distinct difference. DPA is only dependent on whether such a difference exists, not in the sign, + or -, of any given difference.

All an attacker really needs to know in order to mount a successful attack is that it is DES which is being attacked, and that the implementation of DES, at some point, employs a bit which corresponds to a specific output of the S-box, in such a way that its use will affect the power profile samples. The paucity of knowledge required to make a successful DPA attack which completely cracks a hidden DES key on a sealed platform clearly shows that DPA is a very effective means of penetrating a sealed platform.

Only one specific form of DPA attack is described herein, but there are many related forms of DPA attacks which are also possible. Other examples of DPA being used to extract a DES key, which demonstrate the extraordinary power of this technique are presented by:

1. Paul Kocher, Joshua Jaffe, and Benjamin Jun, 1998, *Introduction to differential power analysis and related attacks*, available at http://www.cryptography.com/dpa/technical;
2. Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan, 1999, *Investigations of power analysis attacks on smart cards*, Usenix '99, available at http://www.eecs.edu/~tmesserg/ usenix99/html/paper.html; and
3. Louis Goubin and Jacques Patarin, 1999, *DES and differential power analysis: the "duplication" method,* Proceedings of CHES '99, Springer Lecture Notes in Computer Science, vol. 1717 (August 1999); available at http://www.cryptosoft.com/html/secpub.htm#goubin.

While the effects of a single transistor switching would be normally be impossible to identify from direct observations of a device's power consumption, the statistical operations used in DPA are able to reliably identify extraordinarily small differences in power consumption.

Physical Protection

Physical measures to protect sealed platforms against attack are known to include: enclosing systems in physically durable enclosures, physical shielding of memory cells and data lines, physical isolation, and coating integrated circuits with special coatings that destroy the chip when removed. While such techniques may offer a degree of protection against physical damage and reverse engineering, these techniques do not protect against non-invasive power analysis methods.

Some devices, such as those shielded to United States Government "Tempest" specifications, use large capacitors and other power regulation systems to minimize variations in power consumption, enclose devices in well-shielded cases to prevent electromagnetic radiation, and buffer inputs and outputs to hinder external monitoring.

These techniques are often expensive or physically cumbersome, and are therefore inappropriate for many applications, particularly smart cards, secure microprocessors, and other small, low-cost, devices. Physical protection is generally inapplicable or insufficient due to reliance on external power sources, the physical impracticality of shielding, cost, and other characteristics imposed by a sealed platform's physical constraints such as size and weight.

Software Protection

In contrast to physical protection, smart cards may also be protected from a power analysis attack to an extent, at the software level, by representing data in a "Hamming-neutral" form. The Hamming weight of a binary bit string, such as a data word or byte, is the quantity of bits in the bit string with a value of 1. For example, 10100 will have a Hamming weight of 2, and 1111 will have a Hamming weight of 4. A set of "Hamming-neutral" bit-strings is a set of bit-strings that all have the same number of 1s. If all of the data bytes manipulated by a software application have the same number of 1s, clearly, the power consumed by the device and the noise it emits will not vary as the device processes this data.

For example, one could replace each "1" in a bit string with a "10", and each "0" with a "01". All bit-strings would then have an equal number of 1s and 0s, and theoretically there would be no detectable power or noise variation between any pair of bit-strings. This technique is well known in the art of electrical signalling and hardware design, where it is referred to as power balanced or differential signalling. The benefits of such circuits include:

- reduction in noise emissions or induction of cross-talk in other circuits;
- reduction in ground bounce; because power requirements are constant, the voltage of the ground bus does not rise locally when a circuit switches from low to high; and
- independence from environmental noise; as both electrical lines in a differential pair are influenced by essentially the same level of environmental noise, there is theoretically no net difference detected at the receiving end.

These techniques are commonly used in military, super computer and industrial control applications. Further information on such techniques is widely available, and includes: Kolodzey JS, *CRAY-1 computer technology,* IEEE Transactions on Components Hybrids & Manufacturing Technology, Vol. CHMT-4, No. 2, June 1981, pp.181-6, USA, and Russell RM, *The CRAY-1 computer system*, Communications of the ACM, Vol. 21, No. 1, Jan. 1978, pp. 63-72, USA.

Of course, this approach requires the width of all data buses, memory and computational hardware to be increased to handle the new codings. Using the exemplary mapping above, 0 --> 01 and 1 --> 10, for example, all of these resources would have to double in capacity. More complex mappings are also possible with corresponding increases in overhead, for example, the mapping: 0 --> 0110 and 1 --> 1001, would require a four-fold increase in resource overhead. Since each input bit maps onto its own independent sequence of encoding bits, this method is generally referred to as *bitwise encoding*. Hence, there is a need for Hamming-neutral encoding that does not require such an increase in resources.

The software programming needed to manipulate these Hamming-neutral data bytes can be considerably more complex than regular software programming, requiring the creation of new functions to manipulate such abstract codings mathematically. For example, the boolean calculation (1 OR 0) would map onto (10 OR 01), which could clearly not be effected using the standard OR operator. As well, it is preferable that the new functions perform their calculations in such a manner that the power emitted while calculating would also be Hamming-neutral (referred to herein as Hamming-neutral processing or Hamming-neutral execution), or the benefit of the Hamming-neutral data presentation would be reduced. The overhead of these added hardware capacities and software complexities generally makes the cost of such smart cards too great to be competitive.

Since a normal, unsealed platform is susceptible to attacks potentially more powerful than power analysis (PA), the use of PA in discovery of secret information is primarily directed toward sealed platforms, such as smart cards. However, a simulated power profile of execution can be generated on a simulator for any processor, so it is possible to analyse algorithms for execution on ordinary, unsealed platforms using PA. Hence, although the most urgent need for PA resistance is for use on sealed platforms such as smart cards, PA resistance is required for a much wider variety of platforms.

Improved security is necessary for such devices to be securely used in a broad range of applications in addition to traditional retail commerce, including: parking meters, cellular and pay telephones, pay television, banking, Internet-based electronic commerce, storage of medical records, identification and security access.

There is therefore a need for a method, apparatus and system which reduces the amount of useful information leaked to attackers without resulting in excessive overheads. Reducing leakage refers generally to reducing the leakage of any information that is potentially useful to an attacker trying to determine secret information.

Summary of the Invention

It is therefore an object of the invention to provide a method and system which obviates or mitigates at least one of the disadvantages of the prior art.

One aspect of the invention is broadly defined as a method of decreasing externally observable power modulation from execution of a software program on a computer processor, comprising the steps of: generating a Hamming-neutral set sufficient to span a set of targeted bit strings; and assigning each member of the set of targeted bit strings to a member of the Hamming-neutral set.

Another aspect of the invention is defined as a compiler for compiling high level source code into assembly or machine code, said compiler including software code executable to perform the steps of: generating a Hamming-neutral set sufficient to span a set of targeted bit strings; and assigning each member of the set of targeted bit strings to a member of the Hamming-neutral set.

A further aspect of the invention is defined as a computer readable memory medium for storing software code executable to perform the method steps of: generating a Hamming-neutral set sufficient to span a set of targeted bit strings; and assigning each member of the set of targeted bit strings to a member of the Hamming-neutral set.

An additional aspect of the invention is defined as a carrier signal incorporating software code executable to perform the method steps of any one of generating a Hamming-neutral set sufficient to span a set of targeted bit strings; and assigning each member of the set of targeted bit strings to a member of the Hamming-neutral set.

Brief Description of the Drawings

These and other features of the invention will become more apparent from the following description in which reference is made to the appended drawings in which:

Figure 1 presents an exemplary diagram of a smart card as known in the art;

Figure 2 presents an exemplary physical layout of a system for monitoring and cracking a smart card using power analysis, as known in the art;

Figure 3 presents a flow chart of a broad method of the invention;

Figure 4 presents a flow chart of a preferred embodiment of the invention;

Figure 5 presents an exemplary Hamming-neutral look up table in a preferred method of the invention;

Figure 6 presents the form of a one-dimensional Hamming-neutral address;

Figure 7 presents the form of a multi-dimensional Hamming-neutral address; and

Figure 8 presents a memory layout for Hamming-neutral DES implementation.

Description of the Invention

A method which addresses the objects outlined above, is presented as a flow chart in **Figure 3**. This figure presents a method of decreasing externally observable power modulation from execution of a software program on a computer processor, by performing the steps of:

1. generating a Hamming-neutral set of data sufficient to span a set of targeted bit strings at step **28**;
2. assigning each of the targeted bit strings to a value of the Hamming-neutral set at step **30**; and
3. executing the software program with consideration for the Hamming-neutral set assignment at step **32**, preserving the logic of the software program.

As noted in the Background to the Invention herein above, it has been discovered that theoretically strong cryptographic methods can be cracked very easily by monitoring the noise produced during execution. An easy target for such an attack is a smart card which has very limited resources which can provide protection, and requires external power which provides an easy avenue for power monitoring.

However, power analysis attacks can be used on any manner of software, executing on any manner of microprocessor, micro controller, digital signal processor (DSP), field programmable gate array (FPGA), application specific integrated circuit (ASIC) or the like. Hence, the invention may be useful in many applications.

The invention decreases the magnitude of externally observable information by encoding inputs, internal memory addressing, stored secret keys or other data into a Hamming-neutral form, which minimizes the amount of noise generated during execution. As noted above, the Hamming weight of a bit-string refers to the number of bits with a value of 1 in that bit string. A Hamming-neutral set refers to a set of bit strings which have a like Hamming weight, and hence, the use of a Hamming-neutral set of data will not modulate the power consumed by a device, or the noise it generates.

Known techniques for Hamming-neutral encoding result in a major increase in the necessary hardware registers, buses, or locations on a computer, which have a fixed data width in bits (certain unusual architectures excepted). As noted in the Background, a simple mapping of 0 --> 01 and 1 --> 10, for example, will require a doubling of all of these resources, correspondingly, a more complex mapping of: 0 --> 0110 and 1 --> 1001, would require a four-fold increase in resource overhead. Such mappings can be described as *bitwise* mappings.

The method of the invention differs in that mappings are performed in a *bitstring* manner rather than this *bitwise* manner. That is, rather than mapping each individual bit onto a new coding which at least doubles the width of all resources, the invention maps *groups of more than one bit together* onto new Hamming-neutral codings. This results in far more efficient use of resources, and does not require as great an increase in the width of resources.

For example, a Hamming-neutral set of 8-bit strings with exactly four bits having a value of 1, will have 70 members. Therefore, one can encode any 6-bit string onto this 8-bit set, since $2^6 = 64 < 70$. The Hamming-neutral encodings known in the art increase the width of resources by ratios of at least 1:2, while in this example, the invention has a ratio of 6 bits (unencoded) to 8 bits (encoded), or 1:1.3

As well, the Hamming neutral mappings known in the art, such as 0 --> 01 and 1 --> 10, or 0 --> 0110 and 1 --> 1001, only protect the data with two encodings (one for the 0 bits and one for the 1 bits). In contrast, the method of the invention uses a separate encoding for each bit string, making it far more difficult for an attacker to obtain any useful information. For example, any hardware asymmetries causing bit values or transitions at some bit-positions to have more effect on power consumption than at other bit-positions, have less effect when the instant invention is employed, because it distributes information for any given bit-position in the original algorithm (prior to application of the instant invention) over more bit-positions in the resulting algorithm (after application of the instant invention). The exemplary 6-bit string, for example, uses 64 encodings.

The Hamming-neutral set generated at step **28** must span the set of targeted data, that is, it must have enough members to have at least one entry for each member in the set. Methods for determining the necessary size of the Hamming-neutral set, and how to generate it, are described herein after. Once generated, the members of this Hamming-neutral set may then be mapped onto input bit strings in a one-to-one correspondence at step **30**.

The use of a one-to-one correspondence results in the smallest Hamming-neutral set, which will have the smallest impact on the system resources. However, it is generally preferable that this mapping be performed on a one-to-many correspondence, that is, a member of the target data set may map onto more than one member of the Hamming neutral set. This will make decoding by an attacker even more difficult as the observed correspondence between the target data set and the Hamming-neutral set will not be completely consistent. Note that care must be taken when performing the one-to-many mapping, not to overlap the definitions.

Once the targeted data has been mapped onto a Hamming-neutral set, standard software functions and commands may not operate properly. It is therefore necessary to make whatever modifications are necessary to the software program for it to execute in a manner that preserves the logic of the software. A description of the preferred manner of effecting these changes is provided hereinafter, though various extensions and variations would be clear from the teachings herein. The specific changes, of course, depend on the Hamming-neutral mapping and on the functions involved.

Though functions acting on such data would generally have to be modified, there are many applications of the invention which would not require Hamming-neutral calculations to be performed on the Hamming-neutral data, such as personal data which is merely stored or transferred, and static lookups to memory. When indexing memory addresses, data is stored or manipulated, but is not generally processed or altered. In the case of smart cards, the invention may be used to encode the secret key stored on the smart card, so its value cannot be deduced by power analysis during execution.

To summarize, the bit string Hamming-neutral encoding of the invention:

1. provides Hamming-neutral encoding which is less demanding of system resources than bit wise encoding known in the art;
2. results in a far greater number of encodings which must be deciphered by an attacker;
3. provides a software based solution which is platform independent, in that it can be applied to a wide variety of platforms;
4. can be applied to various components of the targeted code including, for example: addressing, indexing, stored data or input data, critical applications possibly including all of these encodings; and
5. can be augmented with other techniques described hereinafter, including: fixed prefixes and suffixes, parity bits, Hamming-neutral assemblies, asymmetric implementations, and alphabets.

A more detailed description of the invention now follows.

Hamming-Neutral Sets

Let *S* be a set of bit-strings. The set *S* exhibits Hamming-neutrality, or is a Hamming-neutral set, if it has the following properties:

1. $|S| > 1$, where $|S|$ denotes the number of elements in a set *S*;
2. there exists an integer $w > 1$ such that, for every bit-string $x \in S$, $|x| = w$, where $|x|$ denotes the length of a string *x*. That is, all of the bit-strings in the set *S* have the same number of bits (including leading zeros); and
3. there exists an integer $h > 0$ such that, for every bit-string $x \in S$, the number of 1-bits in *x* is *h*. That is, each bit-string in the set *S* has an equal number of bits with a value of 1.

Elements of a Hamming-neutral set are all identical in zero or more bit-positions, whereas two or more elements differ at two or more bit-positions. The bit-positions which are identical for all elements in the set, will be referred to herein as the *fixed* bit-positions, and the bit-positions which differ between elements in the set, the *varying* bit-positions. For example, the set *S* = {1010110, 1011001, 1010101} is a Hamming-neutral set of three elements, all of which are bit-strings of length seven. The fixed bit-positions are the leftmost three, and the varying bit-positions are the rightmost four.

If a Hamming-neutral set *S* is converted to a set *T* by inserting a parity bit in each member of *S*, then *T* is also a Hamming-neutral set provided all of the parity bits are identical. For example, appending odd parity in *S* yields the Hamming-neutral set *T* = {10101101, 10110011, 10101011} of three elements, all of which are bit-strings of length eight. The fixed bit-positions are the leftmost three and one rightmost; the rest of the bit-positions are varying.

However, use of multiple parity bits which contain parity for different selections of bit-positions, as in error correcting code (ECC), may convert a Hamming-neutral set into one which is not Hamming-neutral. Hence, it is necessary to consider how parity is used on a particular platform in determining whether and where Hamming-neutral sets can be employed on that platform.

For the purpose of the present discussion, it may be assumed that either no parity, or only single-bit even or odd parity, is used, so that any sets of values at a site (that is, in a register, on a bus, or in a location) remain Hamming-neutral whether or not any parity bit is included in the value at that site.

Hamming-neutrality is significant to power analysis resistance because:

1. minor asymmetries of hardware implementation aside, elements of a Hamming-neutral set cannot be distinguished by leakage of Hamming weight information, as they all have the same Hamming weight; and
2. minor asymmetries of hardware implementation aside, when all of the bits of an element of a Hamming-neutral set are transitioned to a specific state (that is, when all bits are transitioned to 0's, or when all bits are transitioned to 1's), then the power signature of this action is identical to the power signature which results when this is done to any other member of the set, since exactly the same number of bits are changed and exactly the same number of bits remain unchanged. Hence, transitional leakage for such operations cannot yield information which could help to distinguish elements of a Hamming-neutral set.

As noted in the items above, asymmetries in the hardware implementation may make power consumption more sensitive to the state, or transitions, of some bits than others. This effect is likely to be minor, but can be guarded against if required. For example, if the implementation is more sensitive to the states and transitions of the high-order and low-order bits in a register than to those in between, one can restrict the Hamming-neutral implementations used to those which fix the first and last bit, and vary only the intervening bits.

In general, one can handle the asymmetric implementation problem by dividing the bits into groups with different sensitivities, and ensuring that, within each group of bits with identical sensitivities, the number of bits set is constant within a given Hamming-neutral representation. As input to this technique, one would need to determine the sensitivities at various bit positions. This may be done for example, by a series of hardware measurements on the target platform.

Size of Hamming-Neutral Sets

The number of ways one can choose a subset of *k* elements from a set of *n* elements is the binomial coefficient ${}_n\mathbf{C}_k$. ${}_n\mathbf{C}_k$ is read as "*n* choose *k*", and is defined as ${}_n\mathbf{C}_k = n! \,/\, (k!\,(n - k)!)$ for positive integers *n* and *k* where $n \geq k$.

Let *S* be a Hamming-neutral set with elements of bit width *w*, where the elements have *m* fixed bit-positions and *n* varying bit-positions (so that $w = m + n$), and all elements of *S* have exactly *h* 1-bits. Therefore, there exists an integer *k*, where $k > 1$, such that each element of *S* has exactly *k* 1-bits in its varying bit-positions. This yields:

$|S| \leq {}_n\mathbf{C}_k$, where $|S|$ is denotes the number of elements in a set *S*.

If $|S| = {}_n\mathbf{C}_k$, then *S* may be described as a maximal Hamming-neutral set. That is, the set *S* contains all possible bit strings with *n*-varying bits, having *k* 1-bits in the varying bit positions.

Enumerating Elements of a Maximal Hamming-Neutral Set

One can enumerate all of the elements of a maximal Hamming-neutral set, *S*, in increasing order as non-negative binary numbers (with the usual convention that high-order bits are on the left and low-order bits on the right), by successively constructing ${}_n\mathbf{C}_k$ bit-strings as shown in **Figure 4**.

First, select the number of variable bit positions, *n*, and the number of 1-bit values, *k*, required for the maximal Hamming-neutral set at step **34**. As noted above, ${}_nC_k$ must be equal to or greater than the number of members in the set of targeted bit strings. Clearly, there are an infinite number of possible *n*, *k* pairings for any given targeted set *S*, though generally one will minimize *n*, to minimize the width of the computer processor and associated resources. However, the width of the resources may already be greater than the minimal value for *n* in order to meet other processing requirements. In such a case n is larger, and nCk may correspondingly be larger, providing freedom to use a 1-to-many mapping from original values to Hamming-neutral set elements, rendering the attacker's job harder.

Next, a set V_k is generated at step **36**, where:

V is the set of all varying bit-positions of *S;* and

V_k is the set of all subsets of *V* with *k* 1-bit elements.

This set V_k may be generated in a number of manners, which would be clear to one skilled in the art, for example:

1. by calculating all possible bitstrings that are n bits long, then selecting those with the desired Hamming weight; or
2. by sequentially shifting the bits with a value of 1, through the available bit positions. For example, if $V = \{4, 5, 6, 7\}$ and $k = 3$, that is, bits 4, 5, 6 and 7 are the varying bits, and three of these must be have values of 1, then the first member of V_k will be the bit string with 3 bits having values of 1 in locations 4, 5, and 6. The first iteration would shift the 1-bit at location 6 to location 7, and the next iteration shifting the 1-bit at location 5 to location 6. This would be repeated until all of the bits have been shifted through the varying bit positions. Hamming-neutral techniques for bit shifting are described hereinafter.

If necessary, the elements of set V_k are then sorted into a sequence, *P*, in decreasing order by the sums of their elements, at step **38**. For example, if $V = \{4, 5, 6, 7\}$ and $k = 3$, then $P = \langle \{5, 6, 7\}, \{4, 6, 7\}, \{4, 5, 7\}, \{4, 5, 6\} \rangle$.

The members of set *S* are then assembled as shown at steps **40** through **46**, each successive member of set *S* being assembled according to the next successive element of *P*. This is done by stepping through the members of set *S* using the test at step **40**, and the incrementation through set *P* at step **46**. Step **42** sets the bits in the fixed positions to their corresponding fixed values. As noted hereinafter, fixed prefix and/or suffix bits may for example, be used to specify memory regions or offsets. At step **44**, the varying bit-positions are then set to a value of 1 in the bit-positions specified by the elements of the current subset in *P*, and to values of 0 in the remaining varying bit-positions.

Referring again to the example of set *P* from above, if *S* has fixed bit-positions {1, 2, 3} with the values 101, then the enumeration of the elements of *S* would be the sequence:

$$\langle\ 1010111,\ 1011011,\ 1011101,\ 1011110\ \rangle;$$

since bit positions are numbered from left to right.

A number of terms will now be defined which will aid in the discussion of the techniques which follow.

Population, Spread, and Occupancy

Let $H = \{S_1, S_2, S_3, \ldots, S_r\}$, where $r > 0$, be a set of pairwise disjoint Hamming-neutral sets such that every bit-string in every member of *H* has the same length, *w*. *H* is pairwise disjoint if and only if every pair of distinct elements has an empty intersection; that is, for any *i* and *j* such that $i \neq j$, $S_i \cap S_j = \varnothing$. Such a set *H* is referred to herein as a Hamming-neutral assembly.

That is, all the members of a Hamming-neutral set will have the same Hamming weight. A Hamming-neutral assembly is made of one or more Hamming-neutral sets, each Hamming-neutral set having a different Hamming weight. Therefore, there is no overlap between the different Hamming-neutral sets.

For a Hamming-neutral assembly, *H*, the population of *H* is defined to be:

$$|S_1| + |S_2| + |S_3| + \ldots + |S_r|$$

that is, the total number of elements in all of the sets in *H*, because no two elements are the same.

The spread of *H* is defined to be:

$$H_{max} - H_{min} + 1$$

where H_{max} and H_{min} are the maximum and minimum values, respectively, of elements of members of *H*, when the elements are conventionally interpreted as non-negative binary integer values. The occupancy of a Hamming-neutral assembly, *H*, is defined to be:

(population of *H*) / (spread of *H*)

The occupancy is the percentage of available bit strings in a certain range, which are members of the given Hamming-neutral assembly. For example, if $H_{max} = 127$, $H_{min} = 64$, and the Hamming-neutral assembly has 16 members, then the occupancy would be 16/64 or 25%.

For a single Hamming-neutral set, *S*, one may define the population of *S* to be the population of *H*, the spread of *S* to be the spread of *H*, and the occupancy of *S* to be the occupancy of *H*, where *H* is the Hamming-neutral assembly {*S*}.

Encoding Using Hamming-Neutral Assembly

Multiple Hamming-neutral sets can be generated for different data sets, such as alphabets. An *alphabet* is a finite, nonempty set, such as the set of ASCII or EBCDIC characters, the set of hexadecimal digits, the set of days of the week, or the set of months of the year.

To define the alphabet comprising the upper- and lower-case English letters and the decimal digits for example, a Hamming-neutral set with a population not less than 62 is required (2 x 26 + 10 = 62). A maximal Hamming-neutral set with elements of length 8 with all bit-positions varying and with four 1-bits per element has a population of 70, hence, it could be used to represent this 62 member alphabet. This allows one letter from the above alphabet to be represented in one byte, with each distinct value being represented by a different member of the Hamming-neutral set.

Now, suppose a targeted alphabet is the union of two alphabets, one comprising the upper- and lower-case letters and the decimal digits as above, and the other being the lower-case Greek letters. Further, suppose that one wishes to avoid leaking distinctions among letters of each of the original alphabets, but that the distinction between first of the above alphabets and the second (lower-case Greek) alphabet is not considered useful to an attacker. In that case, one could use a Hamming-neutral assembly with the first member being the maximal Hamming-neutral set mentioned above, with a population of 70, and the second being the maximal Hamming-neutral set of all 8-bit strings with exactly three 1-bits, with a population of 56. Thus, these two sets can be combined into a single assembly sharing the same 8-bit space, and there will be no conflict.

An arbitrary character from the entire union alphabet can then be represented in one byte. Distinctions between characters from the English-decimal alphabet and the Greek are not protected because they have different Hamming weights, but distinctions within the English-decimal alphabet and within the Greek alphabet are protected.

In fact, the alphabet concept of the invention may be implemented to include many different Hamming neutral sets and assemblies for a single software program.

In the preferred embodiment, each variable will have its own mapping and typically, in each operation/lookup, each operand/index will have its own mapping as will the output.

Hamming-Neutral Execution Methods

Hamming-neutral execution or processing refers to the execution of basic computations and functions without exposing information to power analysis by either Hamming-weight leakage or transition count leakage. As well, Hamming-neutral execution should not leak information about layout of data tables.

It is very difficult to build complex electronic components as many short cuts cause imbalance and preserving balance means doing things the bulky way. This is why the techniques taught by Cray et al. only used simple gates. Kocher et al also show how to build simple gates in the patent application filed under PCT serial no. WO9967766, titled "Balanced Cryptographic Computational Method and Apparatus for Leak Minimization in Smartcards and other Cryptosystems", which results in a bulky implementation. The method of the invention, using a table lookup, is far more powerful and flexible than those techniques known in the art.

The techniques for Hamming-neutral execution in the manner of the invention, do increase execution time and data storage space. However, in the context of sealed platforms, the overheads they impose are repaid by the protection they provide against power analysis attacks.

From the techniques described herein, it is possible to perform computations such as shifts, additions, boolean, bit-wise boolean, and other operations, in such a way that transition-count leakage and Hamming-weight leakage do not compromise information one wishes to protect.

Mere use of Hamming-neutral data representations and Hamming-neutral addressing of data tables is *not* sufficient to avoid transition count leakage. To avoid transition count leakage of data, addresses, and certain computational operations, one must generally perform computations in accordance with the following general principal:

> If two operations are not to be distinguishable by transition count, then they must have the same transition count. Moreover, the number of 1-bits which transition to 0-bits should be the same for the two operations, and the number of 0-bits which transition to 1-bits should both be the same for the two operations. This is feasible in general, either by use of Hamming-neutral

table-lookups to implement operations, or by careful implementations using combinations of ordinary computational instructions, or by some combination of these two techniques, as will be evident to one skilled in the art.

As noted, the number of transitions that take place during the computation can be kept constant. In traditional devices, the number of transitions is a function of the current and/or previous state(s) of the device, including the parameters of the particular computation. Leakless devices can be designed for which the type and timing of state transitions during each part of a computation are independent of the parameters of the computation.

Performing Operations by Table Lookup

Whenever an operation takes one or more operands whose representations are short, fixed-length bit-strings which use a Hamming-neutral encoding, one can simply create a table with suitable addressing which contains the results for the operation, and index into it by composing a suitable form of Hamming-neutral address, that is, an address from a set of addresses which is a Hamming-neutral set. If the result is to be concealed, one should also use a Hamming-neutral encoding for the data in the table elements. If the operation produces a result which need not be concealed, then the data elements in the table can use an ordinary, non-Hamming-neutral representation.

An exemplary **XOR** (exclusive OR) operation table for a single pair of bit-encoded Boolean values is shown in **Figure 5**. This example presents a simple Hamming-neutral mapping of 0 --> 01, 1--> 10; with a high output (10) only when one of the inputs is high. The inputs of 00 and 11, and the outputs of 00 are shown for completeness, but of course, they would not be used.

Almost any kind of operation can be performed by a table lookup, or a sequence of table lookups, based on this technique. For example, since one can add, subtract, or multiply one digit at a time, using multiplication and addition tables, and since these operations are also sufficient for long division, one can do integer arithmetic in a Hamming-neutral way, so that (as long as one is careful to avoid transition count leakage as noted previously) one can perform integer arithmetic on data without leaking any information about that data to power analysis.

Bit-wise Boolean operations can also be performed using tables. For example, a table whose elements are stored as bytes, is sufficient for doing arbitrary binary masking operations on operands encoded in eight bits, but representing six bits.

Shifting can also be done using a table-driven approach. Since one can do Boolean operations as well, one can perform arbitrary computations using the techniques described herein, including floating point computations. These techniques may not be suited to high-speed computation or operation in minimal memory space, however, they are highly suited to execution which is resistant to SPA or DPA attacks.

In its ordinary form, that is, without use of Hamming-neutral methods, DES encryption or decryption involves only the following kinds of operations:

1. bitwise **XOR** (exclusive OR) operations;
2. selecting and permuting the bits in a string according to a stored table of integers, as in the initial and final permutations, the expansion permutation, and the compression permutation;
3. extraction of a substring within a bit-string; and
4. concatenation of bit-strings.

Bitwise **XOR** operations can be done by table lookup with a table as shown in **Figure 5**, one pair of Boolean operands at a time, so that instead of a 48-bit wide **XOR** one performs 48 individual **XOR** operations, handling one bit-position at a time. Selecting and permuting bits, both for wide **XOR** operations and for other purposes, can also be done by creating appropriate lookup tables.

Therefore, the entire DES operation can be performed using the techniques described herein.

Alternative methods of Hamming-neutral execution now follow:

Example: End-Off Logical Shifts

Consider an example: Using a data encoding in which one replaces 0 by 01 and 1 by 10, one can represent a 4-bit value in one byte. Suppose that the platform only provides a left or right logical shift by one bit-position. One could then represent an end-off, zero-filled left logical shift or a right logical shift of one bit on this value by two left logical shifts or two right logical shifts followed by **OR**-ing in of the appropriate zero representation (01) by **OR**-ing with 01000000 for right-shifting or 00000001 for left-shifting.

However, if this is done naively, there is potential transition-count leakage. For example, for a left shift, the leftmost bit of the 4-bit value is represented by two bits. Hence, depending on the value of the leftmost represented bit, it is either represented by 01 or 10. As it is shifted end-off, the 01 representation would result in no reduction in 1-bits count followed by a 1-bit reduction, whereas the 10 representation would result in a reduction in 1-bits count followed by no reduction. This could produce observable differences which could be exploited to obtain some information about the value being shifted.

To avoid this, one could proceed as follows: *first*, **AND** the byte with 00111111 (or **OR** the byte with 11000000), which will produce the same transition count and the same Hamming weights before and after the **AND** (or before and after the **OR**), whether the value has 01 or 10 at the left, and *then* perform the two shifts. Then no transition count or Hamming weight leakage can help to distinguish the value of the represented bit shifted out of the register.

It would be clear to one skilled in the art of assembly- or machine-level programming, with employment of the above techniques and principals, how to compose subroutines for shifting a represented quantity of any width any number of bit-positions, without leaking information about the value being shifted, other than its width and the area of memory used for holding the value to be shifted.

For example, suppose one needed to encode, in a Hamming-neutral fashion, a 3-position end-off, zero-filled shift of a byte on a machine that only shifts one bit-position at a time. If the value is bit-encoded, its representation would occupy two bytes, and one must actually shift the 16-bit representation end-off, with 01-pair fill, six positions.

Let us call the left (high order) and right (low order) bytes *L* and *R*, respectively. One will use an auxiliary location *X* (say), and proceed as follows:

1. *R* ← *R* **AND** 11000000;
2. repeat six times: shift *R* right one bit-position;
3. *X* ← *L*;
4. *X* ← *X* **AND** 00111111;
5. repeat twice: shift *X* left one bit-position;
6. *R* ← *R* **OR** *X*;
7. *L* = *L* **AND** 11000000; and
8. repeat six times: shift *L* right one bit-position.

This method of computation accomplishes the desired encoded operation and does not leak transition-count or Hamming-weight information about the represented value which is being shifted.

The above method easily extends to arbitrary width shifting operations.

Example: Extracting a Bit-Field

Suppose one has a 12-bit value, and one wants to extract the 2-bit field comprising bits eight and nine (numbering from left to right). In a bit-encoded representation, there would actually be 24 bits, and the bit-field would comprise bits 15 through 18 inclusive (numbering from left to right). Hence the representation would occupy three bytes, and the desired field would be represented in the last two bits of the second byte and the first two bits of the third byte.

If one wanted to extract the field in a form suitable for proceeding to a table lookup, one would extract it as a 4-bit representation with four high-order 0-bits prepended to make a one-byte value. One would do this as follows, calling the bytes *L* (left), *M* (middle), and *R* (right), respectively, and using auxiliary locations *X* and *Y*:

1. $X \leftarrow M$ **AND** 00000011;
2. repeat twice: shift *X* left one bit-position;
3. $Y \leftarrow R$ **AND** 11000000;
4. repeat six times: shift Y right one bit-position; and
5. $X \leftarrow X$ **OR** *Y*.

If one wanted instead to extract the field in a form providing a one-byte bit-encoded representation, one would add the following step:

6. $X \leftarrow X$ **OR** 01010100.

This step prepends the needed bit-encoded representation of the three leading 0-bits (each 0 represented as 01).

If one wanted to produce a longer representation, one would prepend entire bytes containing 01010101.

The method described here avoids transition-count and Hamming-weight leakage of information about the data values being manipulated and the data values resulting from the computations.

Example: Inserting a Bit-Field

Suppose one has a 12-bit value, and one wishes to insert a 2-bit field comprising bits eight and nine (numbering from left to right). In a bit-encoded representation, there would actually be 24 bits, and the bit-field would comprise bits 15 through 18 inclusive (numbering from left to right). Hence, the representation would occupy three bytes, and the desired field would be represented in the last two bits of the second byte and the first two bits of the third byte.

One would do this as follows, calling the bytes *L* (left), *M* (middle), and *R* (right), respectively, with the value to be inserted into the field represented in another byte *V*, and using auxiliary locations *X* and *Y*:

```
1.    X ← V;
2.    Y ← X AND 00000011;
3.    repeat six times: shift Y left one bit-position;
4.    X <- X AND 00001100;
5.    repeat two times: shift X right one bit-position;
6.    M <- M OR X; and
7.    R <- R OR Y.
```

The method described here avoids transition-count and Hamming-weight leakage of information about the data values being manipulated and the data values resulting from the computations.

Hamming-Neutral Addressing

Hamming-neutral addressing is performed by employing selected Hamming-neutral sets or assemblies. Hamming-neutral assemblies are used for sets of addresses which divide into more than one subset, where the distinctions among the subsets need not be protected.

One Dimensional Hamming-Neutral Addressing

A typical construction for one-dimensional Hamming-neutral addressing is shown in **Figure 6**, following the usual convention that high-order bits are on the left and low-order bits are on the right. If the Hamming-neutral addressing is based on a Hamming-neutral set, then for each such address, the varying bit-positions contain the same number of 1-bits. If it is based on a Hamming-neutral assembly, then the varying bit-positions contain different quantities of 1-bits, depending on how many

Hamming-neutral sets of addresses have been mapped onto the same region of memory. Note that the pairwise disjointness of the members of a Hamming-neutral assembly guarantees that storage elements based on distinct sets from the assembly have distinct addresses, that is, there is no possibility of two elements of data being stored in the same place.

The prefix bit-positions **48** contain fixed bit-values which determine the region of memory to be addressed. The use of such prefixes is well known in the art.

The maximum width of the addressed memory region is the *spread* of any underlying maximal Hamming-neutral set or Hamming-neutral assembly. The number of elements which could be stored in the memory region is the *population* of the set or assembly. The fraction of the region which is actually usable for Hamming-neutral addressing is the *occupancy* of the set or assembly. Definitions for *spread*, *population*, and *occupancy* are given herein above.

One may fine-tune the positioning of the variable bits **50** by appropriate selection of the suffix fixed bit-positions **52**, which provide an offset. Often these suffix bits **52** would contain only zeros, since it is often convenient to store an item in b bits in such a way that its first address modulo 2^b is 0 (2-byte items on even boundaries, 4-byte items on modulo 4 boundaries, and so on). The width of the string of suffix fixed bit-positions **52** determines the width, in memory units, of the storage per element. If it is s, then the space provided for each value to be fetched or stored is 2^s memory units. The width of the entire address, that is, the total number of bit positions, is determined by the type of memory to be addressed and the characteristics of the platform.

Plainly, given the ability to do Hamming-neutral shifting and masking as noted above, addresses can be composed in the form of **Figure 6** as required.

Multiple Dimensions

A typical construction for multi-dimensional Hamming-neutral addressing is shown in **Figure 7**. The prefix **48** and suffix **52** fixed bit-positions are as before, with the prefix **48** selecting the region of memory and the suffix **52** an offset.

If d-dimensional indexing is required, then there are d contiguous groups of varying bit-positions **54**, with widths $w_1, w_2, \ldots, w_d$, where each w_i is chosen so that one can find at least n_i distinct index values which fit in w_i bits, allowing representation of a simple table with an ith index range of n_i entries.

Again, using shifting and masking techniques, one will be able to compose addresses of the above multi-dimensional form as needed. Note that care is required so that during the composition, all intermediate results are Hamming-neutral. This is easily accomplished by zeroing the whole address, then adding each component to it using an OR operation.

An Extended Example: Hamming-Neutral Implementation for DES

A way of implementing the invention upon secret keys under the Data Encryption Standard is now described. The Data Encryption Standard (DES), is described in FIPS publication 46-3, available at http://csrc.nist. gov/fips/ and is both described and extensively discussed on pp. 265-294 of Bruce Schneier's *Applied Cryptography,* 2nd edition, ISBN 0-471-11709-9, 1996, John Wiley & Sons.

Application to DES Key Representation

For the sake of simplicity, 56-bit DES keys are represented in this example in bit-encoded form, where 0 is represented by 01 and 1 by 10, rather than in bit-string encoded format. Implementations in bit-string format would follow logically from the description which follows.

Note that this exemplary mapping doubles the storage for a key from seven bytes to 14 bytes. Parity bits are omitted from the representation, since on a smart card, the keys would be fixed data stored in ROM.

S-box Representation

According to the DES standard, an S-box contains 64 4-bit entries. Since the output bits of an S-box are dealt with individually, a bit-encoded representation (such as 0 –> 01 and 1 –> 10 for example) may be used for elements of the S-boxes also. This puts one S-box entry in one byte. Since 8-bit processors are typical for smart cards, this is a convenient representation for smart card implementations.

However, if a bit-encoded representation for the varying bits within the S-box addresses is used, each S-box will consume too much address space. To avoid this, it is preferable to perform a two-stage look up that employs one large access table.

The S-box Access Table

Ordinarily, an S-box index occupies six bits, so its bit-encoded representation occupies twelve bits. This twelve-bit index means that the naive table will consume 4K bytes of memory ($2^{12} = 4096$). In the preferred embodiment, a conversion is performed to reduce the storage space required for this table into 256 bytes.

To do this, one index conversion table (the *S-box access table*) is employed, which serves for every conversion of a bit-encoded S-box index into a Hamming-neutral S-box element address: it is used once each time an element is fetched from an S-box. It is indexed by a Hamming-neutral address in which there are no suffix fixed bit-positions, there are twelve varying bit-positions in the form of such a twelve-bit bit-encoded index, and the prefix bit-positions indicate the region of memory containing this index conversion table. Indexing into this table with a 12-bit bit-encoded index, the addressed data byte is a corresponding 8-bit index containing some arrangement of four 1-bits and four 0-bits. This 8-bit index is then used to look up the actual S-box. Note that each step of this process is Hamming-neutral.

Memory Layout

The memory region in which the conversion table lies may now be considered. **Figure 8** presents an exemplary layout of such a memory region.

The region of memory indicated in **Figure 8** begins on a 4K boundary, that is, on a 2^{12} boundary. This diagram presents regions of memory in terms of blocks of 256 bytes. The first two bits of the index can only be 01 or 10, and the second two bits of the index can only be 01 or 10, thus the last 1K of the 4K region starting at the 4K boundary can be unused. Moreover, the 1K portion which begins the region is unused, and can provide space for four 256-byte S-box representations, and four 256 byte regions beginning with 0100, 1000, 0111, and 1011, are also unused, providing space for another four 256-byte S-box representations. Hence, the entire eight S-boxes, and the conversion table described in the previous section, can all be stored in a 3K region beginning at a 4K boundary with a good deal of space still unoccupied.

In **Figure 8**, S-boxes 1 through 8 appear as S_1 through S_8, respectively. Each S-box occupies only a sparse portion of its 256 bytes, since only 64 of the 256 bytes are actually used to contain bit-encoded S-box entries. Their *occupancy* is therefore 25%.

The S-box access table sparsely occupies four 256-byte blocks, since only 64 out of 1024 of the bytes are occupied by the result of translation from bit-encoded to an ${}_8\mathbf{C}_4$ Hamming-neutral representation. Its *occupancy* is thus 6.25%.

Effect of Applying the Invention

The implementations according to the instant invention are protected against both SPA and DPA by one or more of the following:

1. removal of features or differences in power profiles, both individual and averaged, by use of computational methods which avoid many situations in which power features or differences would otherwise be expected; and
2. removal of differences between averaged power profiles, by use of computational methods which render such profiles statistically neutral, on average, where they would ordinarily be expected to show distinct differences.

With the comprehensive application of the invention, input and output data from S-box lookups, and the incoming operands and results of all **XOR** operations and permutations, bit-selections, and the like, are all concealed. Since all computations will be Hamming-neutral, all executions will have the same number of 1 bits and the same number of 0 to 1 and 1 to 0 transitions. This assures that each power trace is the same (except for the hardware asymmetry). Thus, all aspects of the DES key are concealed against power-analysis attacks.

The techniques provide protection against revealing any or all of: the data, the data addresses, and the code addresses employed during execution.

Combined Execution Methods

Any of the techniques described herein could be combined with any of the Hamming-neutral data encoding techniques of the co-pending PCT Patent Application Serial No. ----------, titled: "Method and System for Resistance to Statistical Power Analysis", including the average-neutral, permuted, or code-padding execution. These techniques could also be implemented with the Hamming-neutral calculation techniques of the co-pending PCT Patent Application Serial No. ----------, titled: "Method and Apparatus for Balanced Electronic Operations". Greater protection is obtained by using more of these methods at the same time.

In addition, the above methods may be combined, individually or severally, with the methods of producing tamper-resistant, secret-hiding software described in the co-pending data flow patent application, United States Patent Application Serial No. 09/329,117, filed June 9, 1999, titled: "Tamper Resistant Software Encoding", the co-pending control flow patent application, United States Patent Application Serial No. 09/377,312, filed August 19, 1999, titled: "Tamper Resistant Software - Control Flow Encoding", and the co-pending Canada Patent Application, Serial No. 2,305,078, filed April 12, 2000, titled: "Tamper Resistant Software - Mass Data Encoding" to provide a still greater range of protection for a program. Different subsets of the above methods may also be used for different parts of the same program to be protected, depending on the degree of protection with which one wishes to provide each different part.

These techniques may also be combined with other security techniques known in the art such as physical protection or noise introduction, though some of the advantages of the invention may be compromised.

While particular embodiments of the present invention have been shown and described, it is clear that changes and modifications may be made to such embodiments without departing from the true scope and spirit of the invention.

It is understood that as attacking tools become more and more powerful, the degree to which the techniques of the invention must be applied to ensure an adequate level of security, will also rise. It is understood, therefore, that the utility of some of the simpler claimed techniques may correspondingly decrease over time. One skilled in the art would appreciate this and apply the invention accordingly.

The method steps of the invention may be embodied in sets of executable machine code stored in a variety of formats such as object code or source code. Such code is described generically herein as programming code, or a software program for simplification. Clearly, the executable machine code may be integrated with the code of other programs, implemented as subroutines, by external program calls or by other techniques as known in the art.

Because some aspects of the instant invention require precise control over instructions used in algorithms and data layouts in memory, the instant invention is most applicable to assembly- or machine-level implementations. It is less applicable to high-level language (HLL) implementation, because compilers for HLLs usually do not provide the programmer with sufficient control over instruction and memory usage to permit the instant invention to be used effectively.

However, it is possible to employ some or all of the techniques of the instant invention in code generation by a compiler for some HLL. Such a compiler could then be employed to generate PA-resistant machine-code or assembly-code from source-code written in the HLL.

There are many uses for software applications which embed and employ a secret encryption key without making either the cryptographic key or a substitute for the cryptographic key available to an attacker. The method of the invention can generally be applied to these applications.

The embodiments of the invention may be executed by a computer processor or similar device programmed in the manner of method steps, or may be executed by an electronic system which is provided with means for executing these steps. Similarly, an electronic memory medium may store code executable to perform such method steps. Suitable memory media would include serial access formats such as magnetic tape, or random access formats such as floppy disks, hard drives, computer diskettes, CD-Roms, bubble memory, EEPROM, Random Access Memory (RAM), Read Only Memory (ROM), optical media, or magneto-optical media or similar computer software storage media known in the art. Furthermore, electronic signals representing these method steps may also be transmitted via a communication network.

The invention could also be implemented in hardware, or a combination of software and hardware including software running on a general purpose processor, microcode, PLAs, ASICs, and any application where there is a need for leak-minimized cryptography that prevents external monitoring attacks.

It will be clear to one skilled in these arts that there are many practical embodiments of the DES implementation produced by the instant invention, whether in normal executable machine code, code for a virtual machine, or code for a special purpose interpreter. It would also be possible to directly embed the invention in a net-list for the production of a pure hardware implementation, that is, an ASIC.

Typically, the methods and apparatuses of the present invention might be embodied as program code running on a processor, for example, as instructions stored on in the memory of a smart card. Where greater security is desired, the code might additionally be signed by a trusted party, for example, by the smart card issuer. The invention might be embodied in a single-chip device containing both a nonvolatile memory for key storage and logic instructions, and a processor for executing such instructions.

It would also be clear to one skilled in the art that the invention need not be limited to the described scope of credit, debit, bank and smart cards. An electronic commerce system in a manner of the invention could for example, be applied to: point of sale terminals; vending machines; cryptographic smart cards of all kinds including contactless and proximity-based smart cards and cryptographic tokens; stored value cards and systems; electronic payment, credit and debit cards; secure cryptographic chips, microprocessors and software programs; pay telephones, prepaid telephone cards, cellular telephones, telephone scrambling and authentication systems; security systems including: identity verification systems, electronic badges and door entry systems; systems for decrypting television signals including broadcast, satellite and cable television; systems for decrypting enciphered music and other audio content (including music distributed over computer networks); and systems for protecting video signals. Such implementations would be clear to one skilled in the art, and do not take away from the invention.

We Claim:

1. A method of decreasing externally observable power modulation from execution of a software program on a computer processor, comprising the steps of:
assigning each member of one or more sets of targeted bit strings, to members of a Hamming-neutral assembly; and
executing said software program in accordance with said Hamming-neutral assignment, preserving the logic of said software program.

2. A method of decreasing externally observable power modulation from execution of a software program on a computer processor, comprising the steps of:
for each one of a group of targeted sets of bit strings:
generating a Hamming-neutral set sufficient to span said one of a group of targeted sets of bit strings, each of said Hamming-neutral sets having a different Hamming weight; and
assigning each member of said one of a group of targeted sets of bit strings to a member of said Hamming-neutral set;
thereby generating a Hamming neutral assembly.

3. A method of decreasing externally observable power modulation from execution of a software program on a computer processor, comprising the steps of:
generating a Hamming-neutral set sufficient to span a set of targeted bit strings; and
assigning each member of said set of targeted bit strings to a member of said Hamming-neutral set.

4. The method of claim 3 further comprising the step of:
executing said software program with consideration for said Hamming-neutral set assignment, preserving the logic of said software program.

5. The method of claim 4 wherein said step of assigning is performed in a one-to-one correspondence.

6. The method of claim 4 wherein said step of assigning is performed in a one-to-many correspondence.

7. The method of claim 4 wherein said set of targeted bit strings comprises a set of addresses.

8. The method of claim 4 wherein said set of targeted bit strings comprises a set of data.

9. The method of claim 3 wherein the ratio of the bit length of said targeted bit strings to the bit length of said Hamming-neutral set is less than 1:2.

10. The method of claim 4, further comprising the step of:
responding to noise being more sensitive to the states and transitions of certain bit positions in a register by:
restricting the Hamming-neutral implementations to bits other than said more sensitive bit positions.

11. The method of claim 10, wherein said step of responding comprises the step of:
responding to noise being more sensitive to the states and transitions of the high-order bits in a register by:
restricting the Hamming-neutral implementations to those other than said high-order bits.

12. The method of claim 10, wherein said step of responding comprises the step of:
responding to noise being more sensitive to the states and transitions of the low-order bits in a register by:
restricting the Hamming-neutral implementations to those other than said low-order bits.

13. The method of claim 4, comprising the steps of generating and assigning a separate Hamming-neutral set for each set of targeted data which need not be distinguished from other sets.

14. The method of claim 4, comprising the steps of generating a Hamming-neutral set comprising a fixed field and a variable field.

15. The method of claim 14, wherein said fixed field comprises a fixed prefix to define a region of memory.

16. The method of claim 14, wherein said fixed field comprises a fixed suffix to define a memory offset.

17. A method of decreasing externally observable power modulation from addressing of indexed data during computation, comprising the steps of:
pre-computing a Hamming-neutral set or Hamming-neutral assembly; and
encoding addresses according to an enumeration of the elements in said Hamming-neutral set or said Hamming-neutral assembly;
thereby providing resistance to discovery of indices in indexed tables.

18. A method of decreasing noise from execution of a software program on a computer processor, comprising the steps of:
assigning each member of a set of targeted bit strings, in a one-to-one correspondence, to a member of a Hamming-neutral set of data, said Hamming-neutral set of data being sufficient to span said set of targeted bit strings; and
executing said software program in accordance with said Hamming-neutral set assignment, preserving the logic of said software program.

19. A method of Hamming-neutral addressing of indexed data during software computation comprising the steps of:
pre-computing a Hamming-neutral set or Hamming-neutral assembly, and
encoding addresses according to an enumeration of the elements in said Hamming-neutral set or Hamming-neutral assembly, using representation in which each address element in the set or assembly consists of:
zero or more fixed prefix bits, selecting a region in memory;
one or more groups of varying bits, one per dimension of indexing; and
zero or more fixed suffix bits, for selecting an offset in said memory;

thereby providing resistance to discovery of indices in indexed tables when called during execution of said software.

20. The method of claim 4 wherein said step of generating comprises the steps of:

generating a maximal Hamming-neutral set by:

calculating the number of bit positions, *n*, and the number of 1-bit values, *k*, required for said maximal Hamming-neutral set, where *nCk* is equal to or greater than the number of members in the set of said targeted bit strings;

evaluating the members of said maximal Hamming-neutral set as the set of all bit strings with *n* bit positions and *k* 1-bit values.

21. The method of claim 4 wherein said step of generating comprises the steps of:

responding to said set of targeted bit strings having both fixed and variable bit positions by generating a maximal Hamming-neutral set, *S*, as follows:

selecting the number of variable bit positions, *n*, and the number of 1-bit values, *k*, required for said maximal Hamming-neutral set, where *nCk* is equal to or greater than the number of members in the set of said targeted bit strings;

generating a set V_k where:

V is the set of all varying bit-positions of *S;* and

V_k is the set of all subsets of V with *k* 1-bit elements;

sorting the elements of V_k into a sequence, *P*, in decreasing order by the sums of their elements;

assembling each successive member of set *S*, according to the next successive element of *P*, by:

setting the fixed bit-positions to their corresponding fixed values, and

setting the varying bit-positions to a value of 1 in the bit-positions specified by the elements of the current subset in *P*, and to values of 0 in the remaining varying bit-positions.

22. The method of claim 4 wherein said step of generating comprises the steps of:

generating a maximal Hamming-neutral set, *S*, by successively constructing ${}_n\mathbf{C}_k$ bit-strings as follows:

defining *V* as the set of all varying bit-positions of *S*, where *S* is *n* bits in length;

defining *k* as the number of 1-bits appearing in the varying bit-positions of each element of *S*;

defining V_k as the set of all subsets of *V* with *k* elements;

sorting the elements of V_k into a sequence, *P*, in decreasing order by the sums of their elements;

creating each successive string according to the next successive element of *P*, such that:

the fixed bit-positions have their fixed values, and

the varying bit-positions have 1-bits in the positions specified by the elements of the current subset in *P*, and 0-bits in the other varying bit-positions.

23. A compiler for compiling high level source code into assembly or machine code, said compiler including software code executable to perform the steps of any one of claims 1 through 22.

24. A computer readable memory medium for storing software code executable to perform the method steps of any one of claims 1 through 22.

25. A carrier signal incorporating software code executable to perform the method steps of any one of claims 1 through 22.

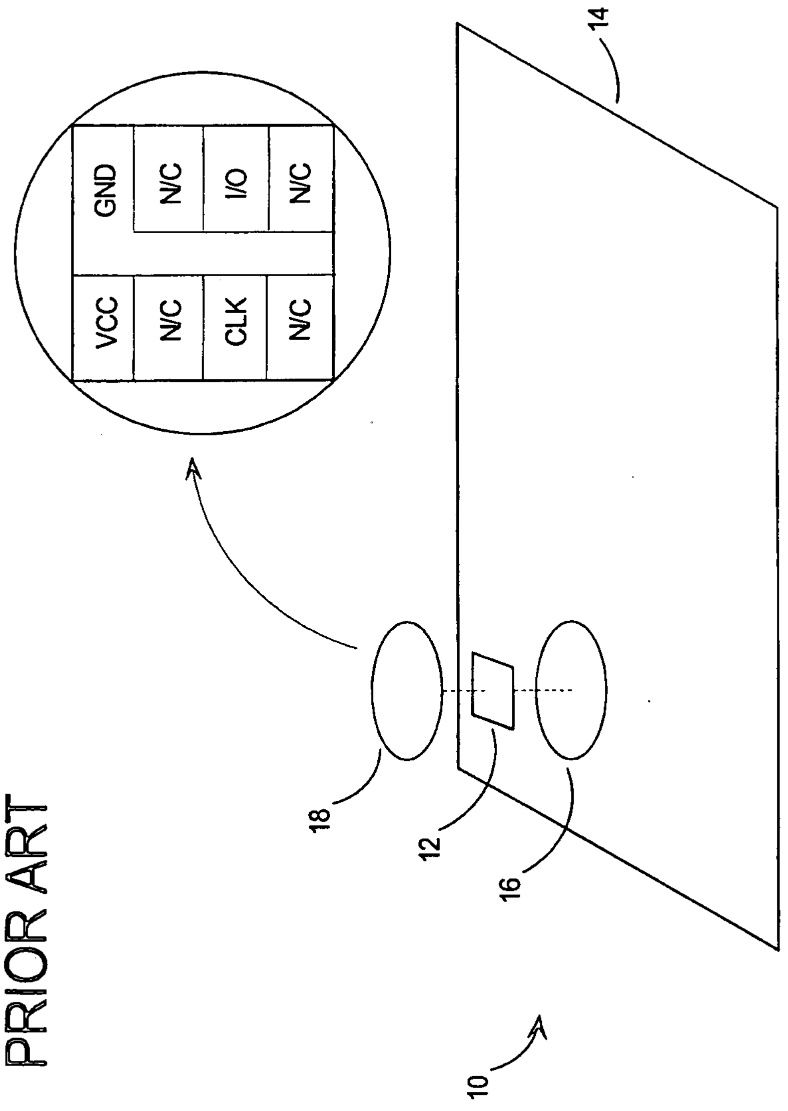


FIGURE 1
PRIOR ART

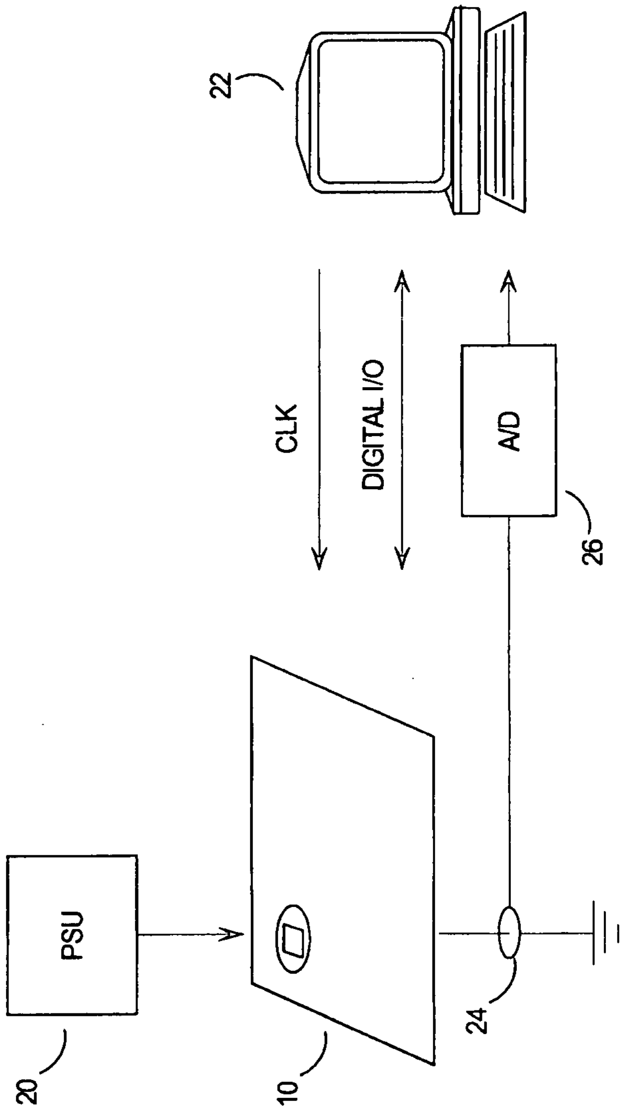


FIGURE 2 PRIOR ART

FIGURE 3

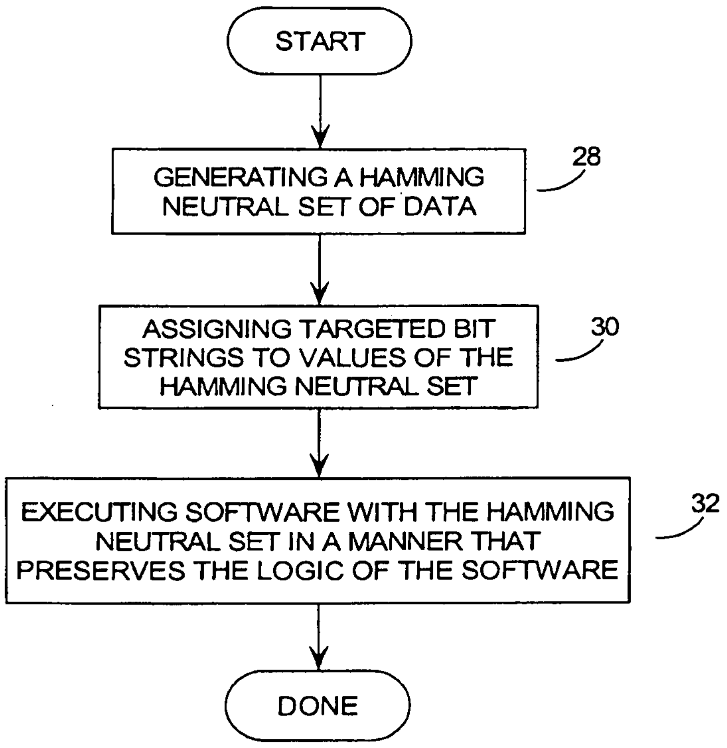

START
GENERATING A HAMMING NEUTRAL SET OF DATA
28
ASSIGNING TARGETED BIT STRINGS TO VALUES OF THE HAMMING NEUTRAL SET
30
EXECUTING SOFTWARE WITH THE HAMMING NEUTRAL SET IN A MANNER THAT PRESERVES THE LOGIC OF THE SOFTWARE
32
DONE

FIGURE 4

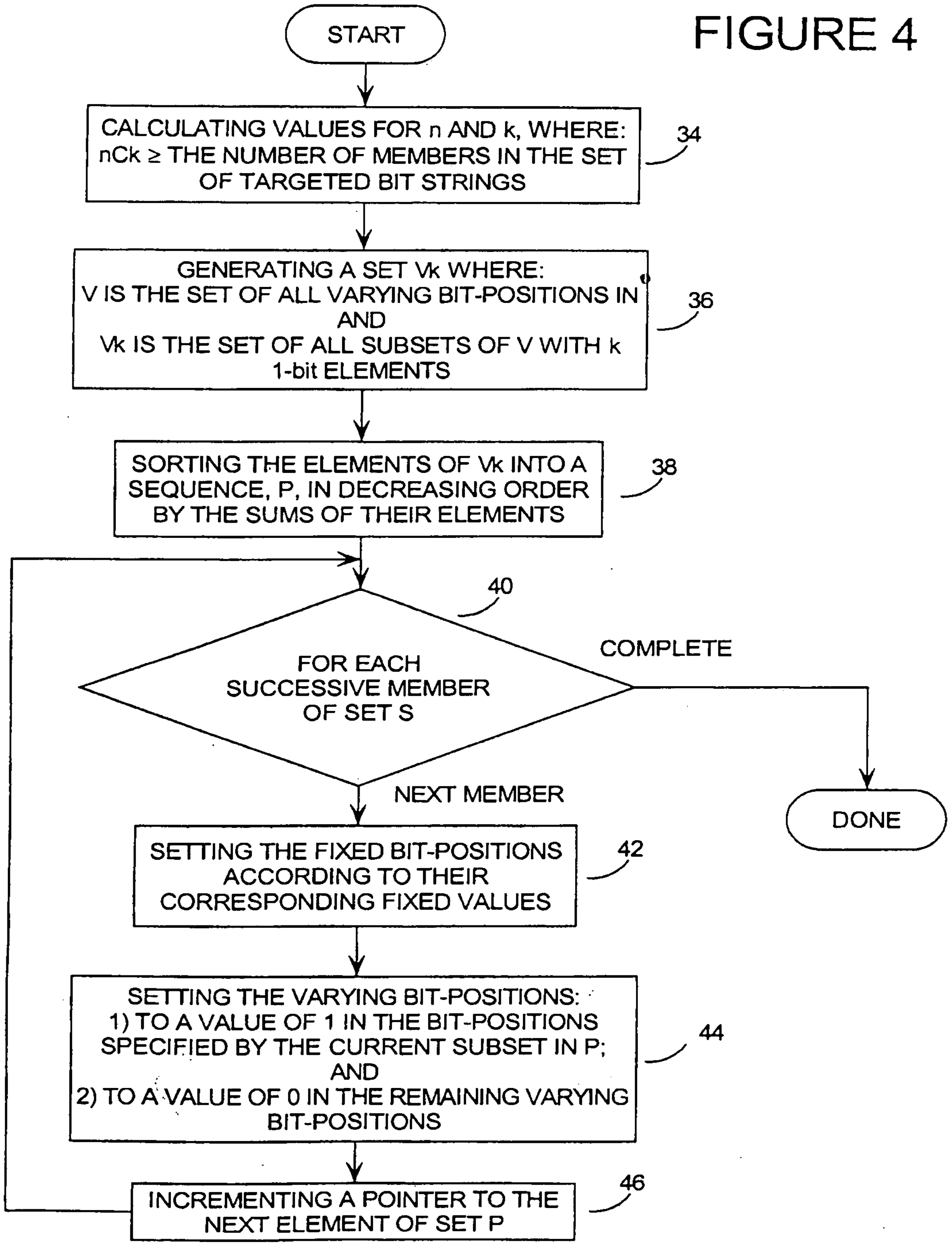

START
CALCULATING VALUES FOR n AND k, WHERE: nCk ≥ THE NUMBER OF MEMBERS IN THE SET OF TARGETED BIT STRINGS
34
GENERATING A SET Vk WHERE: V IS THE SET OF ALL VARYING BIT-POSITIONS IN AND Vk IS THE SET OF ALL SUBSETS OF V WITH k 1-bit ELEMENTS
36
SORTING THE ELEMENTS OF Vk INTO A SEQUENCE, P, IN DECREASING ORDER BY THE SUMS OF THEIR ELEMENTS
38
40
FOR EACH SUCCESSIVE MEMBER OF SET S
COMPLETE
DONE
NEXT MEMBER
SETTING THE FIXED BIT-POSITIONS ACCORDING TO THEIR CORRESPONDING FIXED VALUES
42
SETTING THE VARYING BIT-POSITIONS: 1) TO A VALUE OF 1 IN THE BIT-POSITIONS SPECIFIED BY THE CURRENT SUBSET IN P; AND 2) TO A VALUE OF 0 IN THE REMAINING VARYING BIT-POSITIONS
44
INCREMENTING A POINTER TO THE NEXT ELEMENT OF SET P
46

FIGURE 5

RIGHT OPERAND

LEFT OPERAND	00	01	10	11
00	00	00	00	00
01	00	01	10	00
10	00	10	01	00
11	00	00	00	00

FIGURE 6

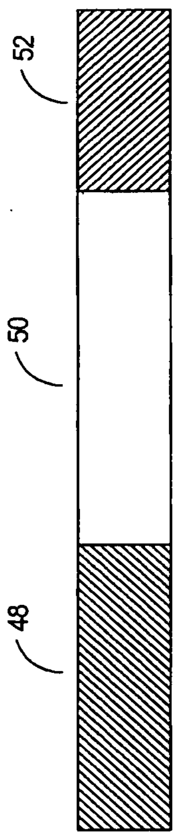


FIGURE 7

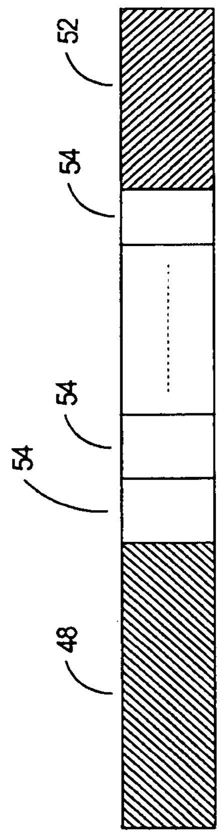


FIGURE 8

2nd BIT PAIR

1st BIT PAIR	00	01	10	11
00	S1	S2	S3	S4
01	S5	S-BOX ACCESS TABLE		S6
10	S7			S8